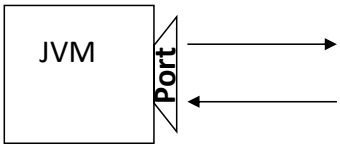
Flux au travers du réseau
Socket et Datagram

Socket en JAVA

Basé sur les protocoles et normes de TCP/IP

Les sockets TCP/IP sont utilisés pour établir des échanges de flux fiables (mode connecté, contrôle d'erreur, renvoi de paquets…), bidirectionnels et point à point entre différentes machines sur Internet (client et serveur).

- Un ***socket*** est un point de terminaison dans une communication bidirectionnelle (via un flux en lecture et un flux en écriture accessible sur la même socket) entre deux programmes fonctionnant sur un réseau.
- Un socket est associé à un numéro de port afin que la couche TCP puisse identifier l'application vers laquelle les données doivent être transmises.



- On peut distinguer la socket :
  - *serveur* qui est une socket d'écoute associé à un port d'écoute. Le serveur attend une demande de connexion de la part d'un client sur ce port.
  - *cliente* qui établit à sa création une connexion avec une socket serveur

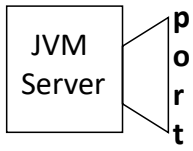
Api Socket en JAVA

- Les packages ***java.net*** et ***javax.net*** fournissent les classes et interfaces suivantes :
  - Adresses IP : ***InetAddress***,
  - Socket TCP :
    - ***ServerSocket :*** attend les demandes de connexion et les accepte ;
    - ***Socket : (client)*** demande l'établissement de la connexion avec le serveur. Classe où l'on trouve les méthodes de communication ;
    - ***JSSE*** (Java Secure Socket Layer),
  - Socket UDP : ***DatagramSocket***, ***DatagramPacket***,
  - Socket MultiCast : ***MulticastSocket***,
  - Classes niveau application (Couche 7 OSI) : ***URL***, ***URLConnection***, ***HttpURLConnection, ….***.

Protocole de mise en œuvre

- Protocole de mise en communication des deux applications par Socket :
  - ***ServerSocket :*** attend les demandes de connexion et les accepte ;
  - ***Socket : (client)*** demande l'établissement de la connexion avec le serveur. Classe où l'on trouve les méthodes de communication ;

- Une application serveur possède un socket d'écoute associé à un port d'écoute. Le serveur attend une demande de connexion de la part d'un client sur ce port.



- Si tout se passe bien, le serveur accepte la connexion.

- Suite à cette acceptation, le serveur crée un nouveau socket associé à un nouveau port.
- Ainsi il pourra communiquer avec le client, tout en continuant l'écoute sur le socket initial en vue d'autres connexions.

Traitement des adresses IP

- La classe ***InetAddress*** représente une adresse du protocole IP :
  - pour les classes ***Socket*** et ***DatagramSocket***.
  - aussi bien par son adresse IP que par son nom d'hôte.

- Les méthodes suivantes permettent la résolution DNS :
  - Obtenir l'adresse ou les adresses de l'hôte dont le nom est passé en paramètre :

```
public static InetAddress getByName(String hostname)
    throws UnknowHostException
```

```
public static InetAddress[]    getAllByName(String hostname)
    throws UnknowHostException
```

// Donne ttes les @ de l'hôte référencées dans le service de noms interrogé

```
public static InetAddress getLocalHost()
```

// pour les adresses locales

Connexion par Socket : Serveur

- Sur le serveur, le processus d'exécution d'une communication par socket en mode connecté est le suivant :
  1. Création d'un ***ServerSocket***,
  2. Attente d'une demande de connexion : ***accept()***,
  3. Récupération des flots d'entrée et de sortie : ***getInputStream()*** et ***getOutputStream()***,
  4. Échange de données avec le client : ***read()***, ***readLine()***, ***write()***, ***printLn()***, ***flush()***,
  5. Fermeture de la connexion : ***close()***,
  6. Attente d'une demande de connexion d'un autre client : ***accept()***.

Création d'une socket d'écoute :

```
ServerSocket serveur = new ServerSocket(port);
```

Attente des demandes et création du socket de communication (qui se fera *via* la classe ***Socket***) :

```
Socket communication = serveur.accept();
//Bloquant en attente de connexion
```

Connexion par Socket : Serveur

public *ServerSocket(**int** port) **throws** IOException*
Crée une socket à l'écoute du port spécifié.

public *ServerSocket(**int** port, **int** backlog) **throws** IOException*
Crée une socket à l'écoute du port spécifié
en spécifiant la taille de la file d'attente des demandes de connexion.

public *ServerSocket(**int** port, **int** backlog, InetAddress bindAddress) **throws** IOException*
idem mais en restreignant l'adresse locale sur laquelle on accepte les connexions.

Principales exceptions :

- ***java.net.ConnectException :*** refus de connexion par l'hôte,
- ***java.net.NoRouteToHostException :*** hôte injoignable,
- ***java.net.ProtocolException :*** erreur de protocole (TCP, ...),
- ***java.net.SocketException :*** erreur de protocole (TCP, ...),
- ***java.net.UnknownHostException :*** erreur de DNS.

Connexion par Socket : Serveur

public *Socket accept() **throws** IOException*
Accepte la connexion et renvoie un nouveau Socket

public void *setSoTimeout(**int** timeout) **throws** SocketException*

Cette méthode prend en paramètre le délai de garde exprimé en millisecondes. La valeur par défaut, 0, équivaut à l'infini.
A l'expiration du délai (accept qui a échoué), l'exception
java.net.SocketTimeoutException est levée et envoyé au client.

public void *close()*
Ferme la socket d'écoute.

public *InetAddress getInetAddress()*
Retourne l'adresse à partir de laquelle la socket écoute.

public *int getLocalPort()*
Retourne le port sur lequel la socket écoute.

Connexion par Socket : Client

- Sur le client, le processus d'exécution d'une communication par socket en mode connecté est le suivant :
  1. Création d'une ***Socket***,
  2. Récupération des flots d'entrée et de sortie : ***getInputStream()*** et ***getOutputStream()***
  3. Échange de données avec le serveur : ***read()***, ***readLine()***, ***write()***, ***printLn()***, ***flush()***,
  4. Fermeture de la connexion : ***close().***

- Création de la socket cliente :

```
Socket Client = new Socket(host, port);
```

- Communication (utilisation des flux ***InputStream***, ***OutputStream*** et de leurs dérivés) :

```
InputStream entree = Client.getInputStream();
OutputStream sortie = Client.getOutputStream();
```


Connexion par Socket : Client

```
public Socket()     //Crée une socket non-connectée
```

public Socket(InetAddress address, int port)throws IOException
Crée une socket de communication et établit une connexion sur le port voulu de la machine dont l'adresse IP est spécifiée.

public Socket (String host, int port)
throws *UnknowHostException, IOException*
idem mais c'est le nom d'hôte qui est spécifié en paramètre.

public *Socket(**InetAddress** address, **int** port **InetAddress** localAddress, **int** localPort)*
throws *IOException*
//Idem au premier et on spécifie en plus une @ locale + port qu'il faut connecter

public *Socket(**String** host, **int** port,**InetAddress** localAddress, **int** localPort) throws IOException*
//idem mais sur les noms d'hôtes

Connexion par Socket : Principales méthodes

Un ensemble de méthodes permet d'obtenir les éléments constitutifs de la liaison établie.

Applicable à une socket ouverte :
maSocket.

InetAddress getInetAddress()
Retourne l'adresse à laquelle la socket est connectée.
L'adresse de type InetAddress concatène le nom de domaine et l'adresse IP correspondante

int getPort()
Retourne le port distant sur lequel la socket est connectée.

InetAddress getLocalAddress()
Retourne l'adresse locale à laquelle la socket est associée.

int getLocalPort()
Retourne le port local associé à la socket.

Fermeture de la socket (et libération des ressources associées) :

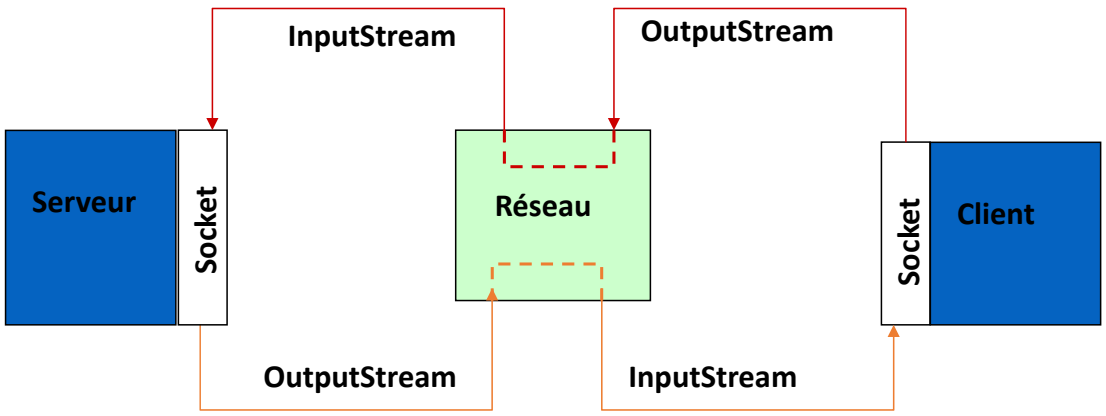
***void** close()*

Communication par Socket : Principes et méthodes

La communication effective sur une socket est basée sur les flux de données ***java.io.InputStream*** et ***java.io.OutputStream***.

***public** InputStream getInputStream() **throws** IOException*
//Retourne le flux d'entrée de la socket

***public** OutputStream getOutputStream()**throws** IOException*
//Retourne le flux de sortie de la socket



Une opération de lecture sur ces flots est bloquante tant que des données ne sont pas disponibles. Cependant, il est possible de fixer un délai de garde pour l'attente des données (similaire au délai de garde pour l'accept du socket serveur).

***public void** setSoTimeout(**int** timeout)**throws** SocketException*

Exemple Simple : le serveur

```
import java.net.*;import java.io.*;

le serveur accepte une connexion, reçoit un entier envoyé
depuis le client et renvoie au client le même entier incrémenté de 1.

public class ServeurBasique
{
    public static void main(String[] args)
    {
        try
        {
            ServerSocket ecoute = new ServerSocket(18000, 5);
            Socket service = (Socket) null;
 while (true)
            {
                service = ecoute.accept();
                OutputStream os = service.getOutputStream();
                InputStream is = service.getInputStream();

                os.write(is.read() + 1);
                service.close();
            }
        }
catch (Exception e) { /* … */ }    }     }
```


Exemple Simple : le client

```
import java.net.*;
import java.io.*;

public class ClientBasique
{
    public static void main(String[] args)
    {
        try
         {
            Socket s = new Socket("localhost", 18000);

        OutputStream os = s.getOutputStream();
            IntputStream is = s.getInputStream();

            os.write((int) 'a');
            System.out.println((char) is.read());
            s.close();
        }
        catch (Exception e) { /* … */ }
    }
}
```